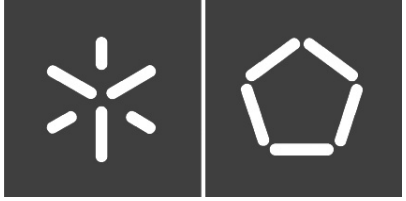
Universidade do Minho
Escola de Engenharia

Manuel Fernando Fortuna Ribeiro Dias

Classificação Automática de Logs

Outubro de 2011



Universidade do Minho
Escola de Engenharia

Manuel Fernando Fortuna Ribeiro

Classificação Automática de Logs

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação de
Prof. Doutor António Luís Sousa

Outubro de 2011

Declaração

Nome: Manuel Fernando Fortuna Ribeiro Dias

Endereço Electrónico: mfd@eurotux.com

Telefone: 967490790

Bilhete de identidade: 13045027

Título da Tese: Classificação Automática de Logs

Orientador: Professor Doutor António Luís Sousa

Ano de conclusão: 2011

Designação do Mestrado: Mestrado em Engenharia Informática

Universidade do Minho, Outubro de 2011

Assinatura: _____

If you show people the problems and you show people the solutions they will be moved to act.

Bill Gates

Agradecimentos

Ao meu orientador António Sousa, pelo estímulo constante e excelente ambiente de trabalho proporcionado.

Ao Professor Paulo Azevedo pelo apoio prestado na análise de soluções para o sistema de previsão.

Aos Colegas da empresa Eurotux, pelas ideias e opiniões.

Ao Pedro pelo companheirismo e pelas ideias pertinentes que contribuíram para esta dissertação.

Aos Familiares, pelo carinho e compreensão nos momentos mais difíceis.

À Gabriela, pelo seu amor e por me apoiar nas minhas decisões.

A todos vós, o meu sincero agradecimento!

Resumo

Data a competitividade característica dos dias de hoje e com o aumento da utilização das infra-estruturas virtuais, ter noção do estado operacional e da utilização de recursos é fundamental, caso contrário o funcionamento do sistema pode ser comprometido.

Os registos de sistema, *logs*, guardam informação sobre o estado e eventos ocorridos. Esta informação indica-nos a carga dos servidores, acessos efectuados, problemas, entre outros aspectos. Actualmente, é necessária a intervenção humana de modo a classificar os *logs* de acordo com a sua criticidade. Se existir um sistema que aprenda os padrões de classificação utilizados, ou que permita a definição de regras, o processo de triagem de *logs* pode ser automatizado.

O principal objectivo desta dissertação é o de explorar métodos que permitam a classificação automática de *logs*. Para este efeito, recorrer-se-á à análise do conteúdo textual dos *logs* e à identificação de padrões de classificação através de técnicas de mineração de dados.

Abstract

In today's competitive world and with the growth of virtualized infrastructures, having the knowledge of operational state and resource usage is critical, otherwise systems can be compromised.

The system records, logs, have information about the state and occurred events. This information allow systems administrator to know about servers CPU load, memory usage, network problems, among other data. Currently, it is necessary human interaction in order to classify system logs according to their criticality. However, if systems could learn about the classification method, or patterns, allowing humans to define classification rules, the whole process could be automated.

The main goal of this dissertation is to explore ways to automatically classify logs, using machine learning in order to identify classification patterns.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Capítulo 1

Introdução

As tecnologias de informação, surgem como meio de automatizar determinadas tarefas fundamentais ao funcionamento da sociedade, tornando–se cada vez mais imprescindível para sua evolução. Actualmente existem sectores económicos dedicados à implementação e manutenção deste tipo de sistemas, que armazenam uma grande quantidade de informação e de serviços. Observando com algum pormenor estes sistemas, conclui-se que são suportados por computadores que efectuam cálculos e aplicam a lógica inerente aos modelos de negócio dos mais variados sectores empresariais.

A disponibilidade destes serviços é fundamental, variando apenas com a criticidade do assunto em questão. Assim, uma empresa de tecnologias de informação que forneça determinado serviço, discute como cliente o nível de disponibilidade que pretende, em percentagem ou em tempo de indisponibilidade, variando o valor monetário de forma proporcional. Considerando, por exemplo, uma empresa cujo modelo de negócio depende das vendas efectuadas num *website* (imagine-se uma bilheteira *online*), o tempo de indisponibilidade do serviço afecta directamente as receitas do cliente, o que não acontece com clientes que apenas possuam o *website* para fornecer informações de contacto.

Nos dias de hoje, para uma empresa da área das tecnologias de informação ser competitiva no crescente mercado deste sector, e de modo a atingir os objectivos de qualidade de serviço e a minimizar o risco de falha contratual, torna-se imprescindível adquirir mecanismos de detecção de problemas.

Para além dos mecanismos referidos, as empresas contam com o apoio da equipa de administração de sistemas, responsável por definir e manter a infra-estrutura de servidores, e com o conhecimento necessário para a resolução de eventuais problemas que surjam.

A equipa é apoiada pelos mecanismos de detecção de falhas, pelo que o seu sucesso está directamente relacionado com a sua eficiência.

1.1 Motivação e objectivos

Diariamente, a equipa de administração de sistemas dedica parte do seu tempo a analisar o estado dos seus servidores, procurando por problemas, ou antecipando problemas latentes nas infra-estruturas, que poderão não acontecer se tratados por antecipação. Para além do tempo que é despendido na procura, é também necessário contabilizar o tempo preciso para descobrir a origem do problema.

De modo a encontrar a origem do problema, por vezes é necessário iterar sobre um conjunto de servidores da infra-estrutura. Na verdade, o que o administrador de sistemas procura são os ficheiros que contém informação sobre o que se tem passado na máquina, isto é, os registos de eventos, também conhecidos por ***Logs***.

O objectivo desta dissertação é criar uma aplicação de suporte aos membros da equipa de administração de sistemas, que agregue a informação dos vários servidores num nó central, minimizando assim o tempo necessário para a detecção e resolução de problemas.

A aplicação criada deve permitir ao utilizadores identificar problemas nos *logs*, e através de técnicas de classificação automática deve tentar automatizar o processo, isto é, classificar automaticamente os *logs* reduzindo o número de *logs* a serem vistos.

1.2 Estrutura da dissertação

Esta dissertação está organizada em cinco capítulos. O presente capítulo, de cariz introdutório, expõe as motivações deste trabalho, traçando as suas metas e objectivos. Apresenta, de forma sucinta, a solução para o problema. Termina com a descrição do relatório em termos estruturais.

O capítulo 2, caracteriza e contextualiza o problema. É apresentado o caso de estudo, e analisados os seus dados. Apresenta-se o método de classificação manual e são exploradas hipóteses para a classificação automática. Por último, são descritas as entidades intervenientes no sistema e é estabelecida a terminologia necessária ao seu entendimento.

O capítulo 3, documenta a concepção do sistema de classificação ETALC[1], seguindo uma cronologia organizada em cinco fases. A fase de agregação, onde se define o repositório de *logs* existente. A fase implementação do processador de *logs*, responsável pela recolha, extracção de atributos e pela aplicação dos mecanismos automáticos de classificação. A fase exploração dos mecanismos automáticos. E por último, a fase de implementação da interface *web* utilizada na classificação e análise de *logs*.

O capítulo 4, analisa as classificações efectuadas pelos utilizadores, bem como os resultados obtidos na classificação automática.

[1] *Eurotux Automatic Log Classification*

O capítulo 5, apresenta as conclusões gerais sobre o trabalho desenvolvido e um conjunto de considerações para trabalho futuro.

Capítulo 2

O problema da análise de *logs*

Os administradores de sistemas geralmente possuem dois grandes tipos de tarefas: estar envolvidos na definição e implementação de novos sistemas, ou na manutenção dos sistemas existentes. No caso da manutenção, é fundamental o conhecimento que o administrador possui sobre o estado do sistema. Isto inclui o conhecimento prático associado aos equipamentos, aos problemas ocorridos no passado, problemas actuais, e aos problemas que poderão surgir. Para além destes, deve estar atento a eventuais falhas de segurança. Para atingir este grau de vigilância, o administrador de sistemas tem ao seu dispor mecanismos que verificam os recursos existentes - *Cacti* [5] e *Nagios* [20] - notificando-o sempre que a saúde do sistema possa ser comprometida. Caso não disponha deste auxilio, deve proceder à verificação manual dos recursos. Para além destes sistemas, tem também disponível outro mecanismo importante, o registo de eventos[1].

O registo de eventos contém mensagens normalmente adicionados pelos programadores com o intuito de fazer a depuração de erros, alertar para problemas, contar o número de ocorrências de determinado evento, entre outros.

De forma a explicar o conceito de comunicação através do mecanismo de registo de eventos, o artigo [23] apresenta o modelo de comunicação definido por Claude Shannon [29], criado para lidar com os desafios existentes na comunicação via radio [7], e estabelece uma analogia entre o modelo proposto e a comunicação via registo de eventos. A Figura 2.1, extraída do referido artigo, ilustra os vários passos existentes.

A comunicação é feita entre o programador que desenvolve a aplicação, e o administrador que faz a manutenção do sistema. O canal de comunicação envolve o envio de mensagens entre as duas entidades, através do registo de eventos. Descrevem-se as entidades intervenientes, sempre neste contexto:

[1] Nos sistema unix o *Syslog* [12] é o protocolo mais utilizado para o registo de eventos.

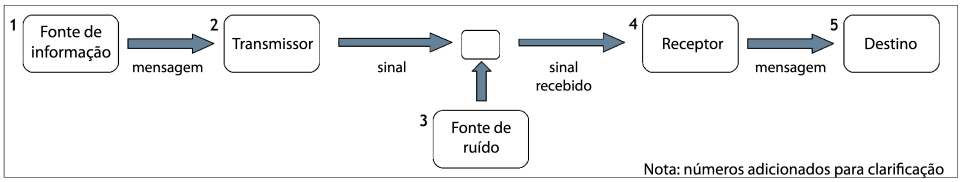


Figura 2.1: Modelo de comunicação definido por Shannon e Weaver [7, 29, 33]

1. **Fonte de informação** - A intenção original do programador, ou de quem faz arquitectura do *software*. Isto é, os intervenientes na edição do código.
   Limitações - competência, memória, decisões efectuadas;
   Estado da informação - rico, processos humanos de pensamento;
   Exemplo - Emitir uma mensagem com informação de estado de determinada operação, que pode ser corrompida por factores desconhecidos.

2. **Transmissor** - A codificação do pensamento efectuada pelo programador, isto é, a representação da ideia do programador descrita numa linguagem de programação.
   Limitações - Capacidade de expressão do programador.
   Estado da informação - Formalizada

3. **Fonte de ruído** - Mensagens não relacionadas com o código criado, bem como qualquer perda de pacotes no canal de comunicação. O nó (quadrado na Figura 2.1) entre os número 2, 3 e 4, representa o registo de eventos.
   Limitações - Informações incompletas e/ou espúria de informação provenientes, por exemplo, da utilização de comunicação através de mensagens UDP[2] frequentemente utilizados em *daemons*.
   Estado da informação - Conteúdo pertinente rodeado por dados que não acrescentam qualquer informação.

4. **Receptor** - Aplicação que filtra as mensagens do registo de eventos para o qual foi concebido. Por exemplo, o *Logwatch* [18] pesquisa mensagens de determinados serviços e apresenta ao receptor o conteúdo já processado.

[2]UDP - *User Datagram Protocol*

Limitações - Filtros estáticos e mecanismos dependentes de tendências, probabilidades e de dados estatísticos.
Estado da informação - Mensagens fora do contexto, propícias a falsos positivos/negativos.

5. **Destino** - Administradores que tomam decisões relativas à manutenção ou alteração do sistema, com base no resultado dos processos de monitorização.
Limitações - Ser humano com outras tarefas que competem pela sua atenção e nível de experiência.
Estado da Informação - Impressões mentais provenientes da observação da informação que passa no processo de filtragem.

O registo de eventos contêm, frequentemente, os únicos dados produzidos ao nível do sistema pelos servidores. Muitos dos sistemas de registo de eventos produzem dados cuja estrutura é bem definida, o que facilita a aplicação de filtros que identificam a informação importante. Outros porém, são excessivamente complexos.

Existem aplicações que se dedicam à síntese destes registos, por exemplo o *Logwatch* [18] e o *Logcheck* [17], facilitando o processo de pesquisa - *Receptor* no modelo de comunicação da Figura 2.1. Uma forma de extrair a informação necessária do conteúdo dos *logs* é através do uso de expressões regulares. Neste método, as aplicações possuem listas com padrões de comportamento correcto e incorrecto, e vão classificando os registos de acordo com estas listas [23]. Outra é através de métodos estatísticos com o sugerido em [16].

Existem aplicações que seguem a filosofia da aplicação de filtros, como é exemplo o *LogWatch* [18], o *Logcheck* [17], o *System Event Log*, etc. Existem também soluções comerciais, como por exemplo o *NMS* [22] e o *Splunk* [30].

A qualidade do resultado destes sistemas automáticos de classificação depende da qualidade dos recursos de entrada. É aqui que as soluções de classificação de *logs* enfrentam o primeiro desafio, uma vez que a alteração de formato pode invalidar os filtros já definidos. Existe uma grande probabilidade de que estas situações se verifiquem, uma vez que os administradores de sistema fazem actualizações de *software* frequentemente, tanto para obter as funcionalidades de versões mais recentes, como para colmatar falhas de segurança. Por exemplo, um filtro automático pode deixar de ter efeito se o sistema operativo alterar o formato das mensagens utilizadas pelo programa.

Tendo em consideração a evolução das metodologias de *logging*, faz com que seja difícil de os interpretar [23]. Os problemas chave são: funcionalidades adicionadas de forma incremental, criadas normalmente pelo programador para fazer depuração de erros; falta de previsão dos programadores de

como os *logs* seriam utilizados pelo administrador de sistemas; e a quase imperceptível confusão entre pressupostos humanos com a informação correcta do *log* [6]. O número esmagador de mensagens por si só é um desafio intimidador em muitos dos casos, bem como a vasta gama de formatos existentes de mensagens, tipos e esquemas [14].

Todos estes factores são prejudiciais para a visão que o administrador tem sobre os seus sistemas.

2.1 Estudo de caso - Eurotux

O sistema de *logs* da Eurotux é um modelo de classificação, passível de ser analisado. O facto da infra-estrutura suportar serviços de vários clientes, utilizando para o efeito diversos sistemas operativos, torna-o num bom estudo de caso, pois cobre um conjunto amplo de serviços. A presente secção refere-se à situação actual do mecanismo de *logging* existente.

Existem algumas fontes de *logs* provenientes de aplicações de síntese de registos de eventos, e outras provenientes de aplicações específicas. Tipicamente, as fontes de *logs* existentes enviam as suas notificações através de email, dando entrada numa caixa de correio partilhada pelos membros da equipa de administração de sistemas, acedida via IMAP[3] que permite que todos consultem as mensagens sem as apagar do servidor de email. De modo a estruturar e facilitar a pesquisa, o processo de entrada de *logs* é controlado, sendo inseridos na directoria do cliente a que pertencem. A Figura 2.2 ilustra a arquitectura do sistema de catalogação e classificação de *logs*.

Para além da catalogação dos *logs* na caixa de correio IMAP, o sistema disponibiliza uma interface *web* através da qual são efectuadas as classificações. Por classificação, é registado na base de dados quem valida o *log*, a classificação atribuída e a respectiva data.

Após a concepção do sistema de classificação, construído decorrer deste estudo, a interface *web* e a base de dados foram substituídas.

2.2 Tipos de *logs*

Existe uma grande diversidade de tipos de *logs*, o que dificulta o processo de classificação, onde cada tipo possui a sua estrutura e conteúdo. Felizmente, no presente estudo de caso são utilizadas aplicações de síntese de *logs*, que agregam e estruturam os dados, reduzindo a diversidade de conteúdos e tipos a analisar. Apesar do ambiente ruidoso, existe um conjunto de tipos principais, que representam a maioria dos *logs*. Este tipo de *logs* provêm na maior parte dos casos de sistemas de execução automática de *scripts*, ou de aplicações de *backup* de dados.

[3] *Internet Message Access Protocol*.

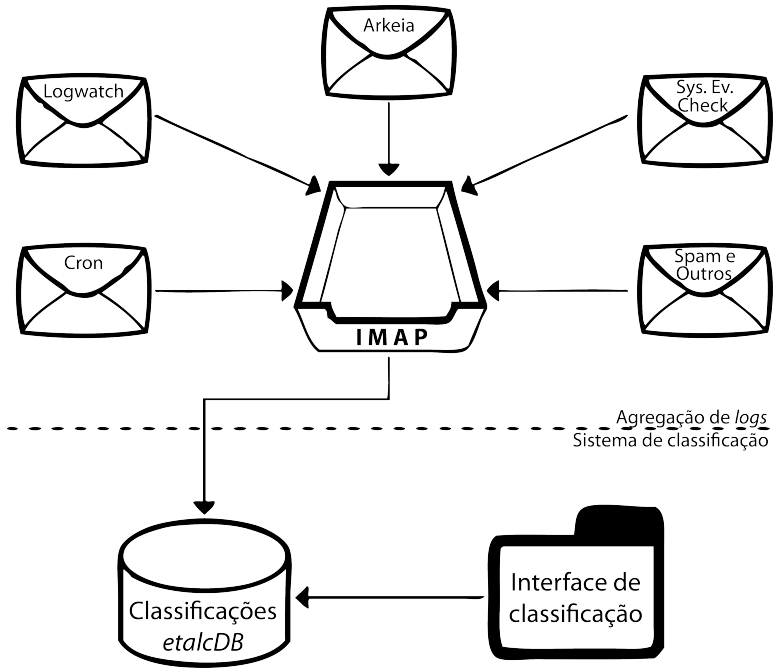


Figura 2.2: Arquitectura do Sistema de Catalogação e Classificação de *Logs*

Nas seguintes secções descrevem-se, sucintamente, as principais fontes de *logs*.

2.2.1 *Cron*

O *Cron* [8] é um escalonador de tarefas associado aos sistemas *Unix*, embora haja implementações para sistemas Windows [15]. Através deste serviço é possível agendar tarefas e definir a sua periodicidade. É utilizado na administração de sistemas e executa *scripts* que efectuam algo específico na máquina, por exemplo, verificar a integridade dos ficheiros do sistema. O resultado da execução dos *scripts* é registado e enviado através de email.

2.2.2 *Logwatch*

O *Logwatch* [18] analisa os *logs* do sistema em ambientes *Unix*. Este serviço personalizável percorre os *logs* do sistema operativo, e ao fim de um dado período de tempo, agrega a informação e regista o resultado. O seu resultado provém da estruturação dos dados contidos nas linhas do *syslog* [12]. Uma vez que a geração de *logs* é regular, a ausência de notificação pode ser problemática.

2.2.3 *System Event Check*

O *System Event Check* é um sistema de notificação de eventos de servidores *Windows*, análogo ao funcionamento do *Logwatch*. A implementação deste sistema de notificação, no presente estudo de caso, é proprietária. Porém, existem alternativas disponíveis publicamente como é o caso do *WindowsEvent* [11], que tem também expressa as suas notificações via email.

2.2.4 *Arkeia*

A aplicação *Arkeia* [4] é uma solução para a salvaguarda de dados de forma global ou incremental, indispensável em acções de *disaster recovery*, e para protecção contra alterações acidentais dos dados. Preserva a estrutura de directorias, *links simbólicos* e quaisquer atributos especiais do sistema de ficheiros. Esta aplicação suporta os diferentes sistemas operativos utilizados na infra-estrutura. Os *logs* gerados contêm informação sobre itens copiados, e possuem e a seguinte característica: se existir um *log* B correcto, mais recente que A com informação de problemas, então os problemas de A podem ser ignorados. Isto é, se existir um *backup* recente que foi concluído com sucesso, então os anteriores com falhas podem ser ignorados.

2.2.5 Outros

Logs gerados por diversas aplicações, como por exemplo o *RSync*[4] [26]. Nesta categoria encontra-se a definição de vários tipos de *logs*, por aplicação. Como o sistema de notificação envolve o envio de emails, na caixa de correio também é possível encontrar correio electrónico não solicitado - *spam*.

2.3 Distribuição dos *logs* pelos tipos existentes

O Gráfico apresentado na Figura 2.3 mostra a distribuição dos tipos de *logs* existentes. Os dados foram obtidos através da análise de uma amostra de *logs* proveniente dos servidores que suportam os serviços.

Tendo em consideração o tempo despendido na análise de *logs*, pode afirmar-se que é necessário dar mais atenção às fontes que produzem mais *logs*, nomeadamente na definição de filtros estáticos, abordados na Secção 3.5.1.

2.4 Classificação manual

O processo de classificação é efectuado diariamente pelos vários membros da equipa de administração de sistemas. No cenário perfeito, os *logs* seriam

[4]Aplicação utilizada na sincronização de conteúdos.

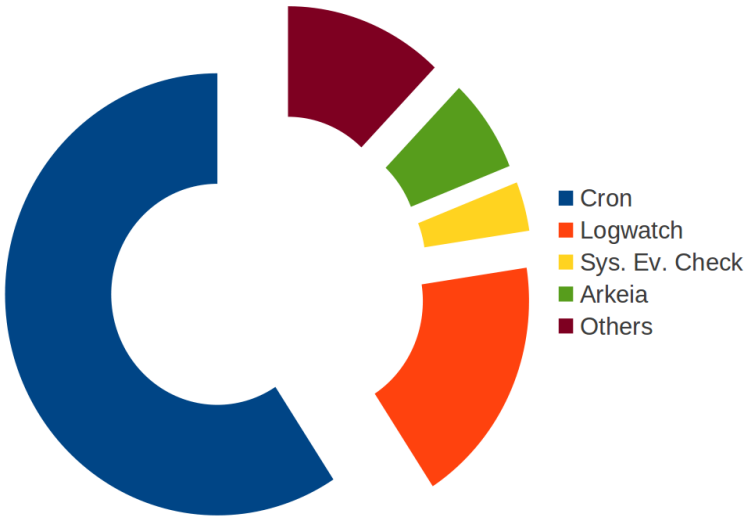


Figura 2.3: Distribuição dos *logs* pelos tipos existentes

classificados assim que dão entrada na caixa de correio de notificações, minimizando o tempo necessário para intervir e resolver o problema. Porém, a teoria não se assemelha à prática, uma vez que esta equipa não se dedica exclusivamente à análise e classificação. Por vezes, devido ao elevado número de *logs* a analisar há problemas que não são detectados, sendo posteriormente descobertos pelo sistema de monitorização de recursos - *Nagios* [20].

Regressando ao cenário ideal, e de modo a garantir que todos os problemas são detectados, cada *log* deve ser visto por dois elementos da equipa de administração de sistemas, bastando que apenas um o classifique como incorrecto para que seja considerado como problema.

Para cada problema encontrado, o administrador de sistemas cria no serviço de *tickets* - *RT*[5] [32] - um registo com a descrição necessária. Posteriormente, a tarefa registada é submetida a um processo de triagem, e é atribuída a um dos elementos da equipa de administração de sistemas.

Para evitar que os elementos classifiquem sempre os mesmos *logs*, a equipa está dividida em vários grupos de dois elementos, que trocam entre si os *logs* produzidos pelas máquinas dos vários clientes. Esta medida visa minimizar a observação de padrões de classificação por parte dos elementos, que os induzem à classificação errónea. Isto é, um elemento da equipa que veja sempre os *logs* de um dos servidores, onde não é usual encontrar anomalias, pode passar a assumir que esta premissa se mantém.

Há também a questão da subjectividade no processo de classificação. Cada utilizador tem a sua forma de classificar, as suas regras. Determinado

[5] *Request Tracker.*

problema pode ser considerado grave por um utilizador, e não tão grave por outro, variando o nível de criticidade atribuída ao problema com nível de conhecimento que o utilizador possui sobre o sistema.

Para além deste problema é necessário considerar que o conteúdo dos *logs* varia, ainda que dentro do mesmo tipo. Por exemplo, nos *logs* gerados pelo *Logwatch*, existe uma secção chamada espaço em disco, onde se encontra uma linha por cada volume de dados da máquina, que indica a respectiva utilização. Neste caso pode haver problemas se o espaço ultrapassar determinada percentagem, porém, esse limite varia de servidor para servidor, e entre os vários tipos de volume[6].

2.5 Classificação automática

Existem diversos métodos utilizados para classificação, como: a definição de filtros estáticos; a aplicação de modelos estatísticos, no qual o *HMM (Hidden Markov Model)* [19] é exemplo; e a utilização de técnicas de mineração de dados. A aplicação de modelos estatísticos estão para além da investigação proposta para esta dissertação, pelo que o assunto não será explorado.

Nas secções seguintes, descrevem-se as técnicas de aplicação de filtros estáticos e de mineração de dados, sempre no contexto da classificação de *logs*.

2.5.1 Filtros estáticos

O método mais simples e determinístico utilizado na classificação são os filtros estáticos, que aplicam regras previamente definidas. Estas regras, são geralmente definidas por expressões regulares, aplicadas ao conteúdo dos *logs*.

Porém, este método têm inerentes alguns problemas, como o facto das regras serem definidas especificamente para determinado tipo de *log*. Dentro do mesmo tipo, as regras podem ainda estar associadas a um cliente como, por exemplo, a definição de uma lista de controlo de acesso. Como o referido no início do capítulo, na actualização ou alteração da topologia da infra-estrutura, o conteúdo dos *logs* pode ser alterado, invalidando os filtros definidos até à data.

Um sistema ideal de classificação automático de *logs* deveria ser capaz de se adaptar às suas alterações, ou pelo menos às pequenas variações de valor no seu conteúdo. Tendo em consideração este conceito, utilizar as técnicas de extracção de conhecimento utilizados nas áreas de Sistemas de Suporte à Decisão e Inteligência Artificial, pode constituir a solução para este problema.

[6] O volume pode ser uma imagem de um DVD, em que a percentagem de espaço ocupado seja 100%. Este caso, não problemático, pode iludir o utilizador ou o sistema de classificação.

2.5.2 Mineração de dados

Armazenar as classificações e problemas existentes nos *logs* é extremamente importante. Estes dados incluem o conhecimento necessário para aplicar as técnicas de mineração de dados existentes, de modo a encontrar os padrões necessários à classificação automática. Na verdade, o que se pretende extrair são informações relevantes e desconhecidas que facilitem o processo.

A extracção de conhecimento é indicada neste estudo de caso, visto que os *logs* são únicos. Isto implica que o sistema de classificação seja suficientemente genérico, e tente encontrar com boa precisão se determinado *log* pertence há categoria dos que possuem informação sobre problemas e, caso afirmativo, classificar o grau do problema. Por outro lado, a disponibilidade de um grande conjunto de dados é ideal, tanto para o treino dos algoritmos de classificação, como para efeitos de teste.

Na mineração de dados existem várias áreas de actuação, nomeadamente para classificação, regressão linear, segmentação, previsão e descoberta de regras de associação [34]. Das diversas áreas, a que melhor se ajusta é a de classificação. Isto porque, quando um *log* é analisado por um membro da equipa de administração de sistemas, é integrado no grupo dos *logs* sem problemas (com criticidade 0), ou no grupo dos que têm informação sobre problema(s) (com criticidade variável entre 1 e 5). Dito de outro modo, observa-se que o resultado da classificação é um valor nominal, pertencente ao conjunto dos números inteiros existentes no intervalo de 0 a 5.

O processo para extrair conhecimento de dados é definido na literatura em [28]. Os autores enumeram cinco fases, sendo: selecção; pré-processamento; transformação; *data-mining*; interpretação. A Figura 2.4, retirada do livro referido, ilustra as fases existentes.

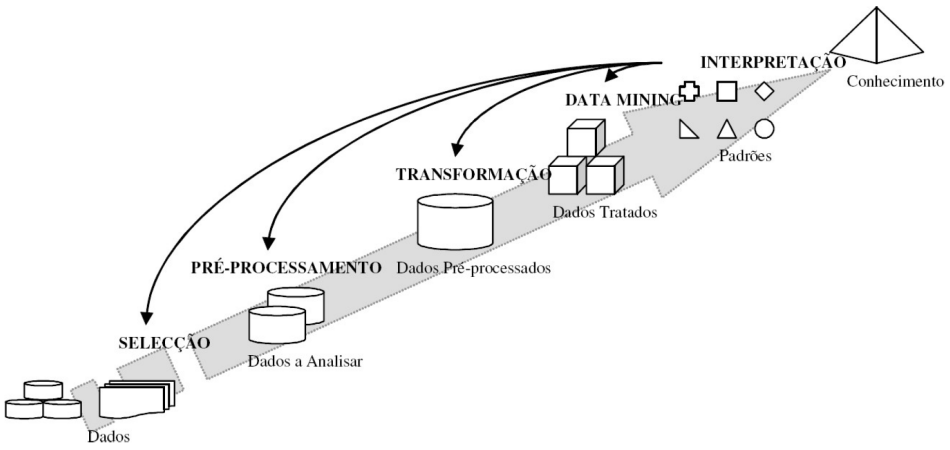


Figura 2.4: Fases utilizadas na extracção de conhecimento [28]

As secções seguintes descrevem sucintamente cada fase, no contexto da classificação de *logs*.

Selecção

Os dados provêm de fontes heterogéneas e nem sempre constituem utilidade para a resolução do problema, por isso é necessário escolher os dados a utilizar na extracção de conhecimento. Recordando a Figura 2.2, é necessário recolher os *logs* que efectivamente contribuem para o treino dos algoritmos, filtrando correio electrónico não solicitado, ou inclusive *logs* de aplicações que não estão em uso.

É nesta fase que se extraem os atributos que constituem a base de conhecimento, ou seja, é necessário identificar o que caracteriza os *logs* com problemas. Esta informação deve ser obtida acompanhando a equipa de administração de sistemas, percebendo quais os critérios de classificação utilizados. Note-se que a qualidade das classificações automáticas depende, em grande parte, dos atributos escolhidos. Utilizando o sistema de *backup* como exemplo, alguns do possíveis atributos a extrair seriam: a hora de início, a de fim, e o volume de dados transferido.

Pré-Processamento

A fase de Pré-Processamento envolve o limpar da informação reunida, eliminando ruído, erros e omissões de valor. Para além deste procedimento, também pode ser necessário proceder à normalização dos atributos escolhidos - por exemplo, discretizar o atributo *data* contido no *log*.

Salienta-se que esta fase representa a maioria do esforço necessário para a extracção de conhecimento.

Transformação

Na fase de transformação, os *logs* são organizados por tipo, dando origem a repositórios de dados organizados e de grande dimensão. A ideia é poder separar os dados de modo a permitir aplicar algoritmos diferentes. Por exemplo, separar os *logs* do *Logwatch* dos restantes, uma vez que podem ser divididos por serviços. Apesar de tal se verificar no sistema de classificação implementado, este assunto não vai ser explorado, uma vez que esta dissertação se resume à classificação genérica de *logs*.

Data-mining

Os algoritmos de mineração de dados são aplicados aos dados, de forma a extrair o conjunto de regras existentes. Nesta fase, a finalidade é seleccionar os métodos e técnicas necessários para a classificação de *logs*. Alguns

exemplos de algoritmos a aplicar são: Arvores de Decisão, Redes Neuronais Artificiais ou a Classificação *Bayesiana*.

Interpretação

Os padrões identificados na fase de mineração são aplicados na classificação de novos *logs*. Tipicamente, quando o conjunto da dados utilizado possibilita, é criado o conjunto de dados de teste. Estes dados são os *logs* com a respectiva classificação. O que se pretende é testar o sistema de classificação, comparando o resultado obtido com a classificação previamente efectuada pela equipa de administração de sistemas. Deste modo, é possível obter a precisão do sistema, dividindo os casos em que o resultado coincide com a classificação pelo número total de casos.

Se se verificar que a precisão não é a pretendida ou pode ser melhorada, pode haver necessidade de retornar às fases anteriores. Pode-se, por exemplo, alterar os atributos em uso, o número de exemplos de treino ou ainda trocar os algoritmos utilizados.

Representação dos dados

Existem vários métodos para a representação da informação resultante da fase de extracção de conhecimento [34] - fase de *Data-Mining*.

As tabelas de decisão representam a forma mais simples de estruturar a informação. Na verdade, esta forma de conhecimento é conseguida através da construção de uma tabela, onde às colunas se faz corresponder o valor dos atributos seleccionados e a respectiva classificação atribuída. Na sua utilização, o sistema automático de classificação consulta a tabela, tentando fazer corresponder o valor dos atributos do novo *log* com os valores das várias entradas. Caso seja encontrada uma entrada com valores idênticos, é retornada a classificação atribuída.

As tabelas de decisão não são solução para o problema da classificação de *logs*, uma vez que frequentemente o valor das variáveis em análise é contínuo, o que implicaria discretizar o seu valor e incluir uma entrada na tabela para cada caso[7].

As árvores de decisão são outro estilo de representação. Cada nó da árvore constitui um teste a determinado atributo em particular, onde tipicamente o seu valor é comparado com uma constante. Por vezes, os nós comparam dois atributos, ou utilizam uma função de um ou mais atributos. Nas árvores de decisão, as folhas indicam a classificação a atribuir às instâncias que as atinjam, isto é, que cumpram os requisitos necessários para atravessar o caminho entre o nó raíz e a folha, passando pelos testes efectuados pelos nós. Caso o atributo a testar seja numérico, o nó normalmente

[7]Por exemplo, o número de linhas do conteúdo do *log*.

determina se o valor é maior ou igual a um valor constante pré-definido, resultando na divisão da árvore em dois ramos[8] - como, por exemplo, o teste ao atributo número de linhas, maior ou igual a 1000.

Existem outras formas de representação do conhecimento extraído da aplicação de algoritmos de mineração de dados, que não são exploradas nesta dissertação. Dois exemplos são as regras de classificação e as de associação.

2.6 O problema da classificação de texto

A mineração de dados é também utilizada para a classificação de documentos [9], tendo por base o seu conteúdo, inserindo-os em categorias predefinidas e facilitando pesquisas posteriores. No processo de selecção de atributos, cada documento é representado por um vector de palavras [27] (em que a cada índice se faz corresponder uma palavra, cujo valor é o número de ocorrências). Por motivos de eficiência e eficácia, a redução do número de atributos é importante. Para conseguir atingir este objectivo, reduzem-se os atributos/palavras, às que surgem com mais frequência nos documentos, e depois seleccionam-se os que melhor caracterizam as categorias. Vários métodos para selecção de atributos são comparados em [35].

À primeira vista, a utilização de classificadores de texto pode parecer ideal para a categorização de *logs*. Porém, a afirmação não poderia estar mais longe da verdade. O processo em si, pode ser semelhante, nomeadamente o pré-processamento e transformação de dados, e a aplicação de algoritmos de classificação. No entanto, a selecção não pode ser efectuada da mesma forma. Isto porque na selecção de texto, é possível deduzir a que categoria pertence o documento observando a frequência com que aparecem determinadas palavras-chave que são específicas de uma categoria em particular. Já nos *logs*, o conjunto de atributos chave não pode ser extraída directamente do texto, mas através das regras sobre as quais os vários elementos da equipa de administração de sistemas se regem. Reforçando a afirmação, ter um vector que representa o conteúdo de um *log* e que indique o número de ocorrências de um conjunto de palavras não constitui qualquer informação sobre o problema. Como solução, poder-se-ia considerar um vector, onde às dimensões se fizessem corresponder os atributos que efectivamente possuem informações sobre o estado do servidor - por exemplo, a percentagem de espaço ocupado em disco, o tempo de execução de determinado processo, a origem do *log*, e outros.

2.7 Plataformas para extracção de conhecimento

Actualmente, existem algumas ferramentas que podem ser utilizadas na extracção de conhecimento para a classificação. Todas, implementam os algo-

[8] Existem formas alternativas para teste de valores numéricos descritos em [34].

ritmos típicos, utilizados na mineração de dados.

O *Weka*[9] [34] agrega vários algoritmos dos diversos paradigmas/abordagens da inteligência artificial. Procede à análise computacional e estatística dos dados fornecidos, recorrendo a técnicas de mineração de dados, tentado indutivamente a partir dos padrões encontrados gerar hipóteses para soluções.

O *RapidMiner* [25] é análogo ao *WEKA*, sendo uma ferramenta *open-source* bastante utilizada.

O *Apache Mahout* [3] é uma biblioteca para extracção de conhecimento escalável, suportando grande volume de dados. De modo a conseguir atingir este objectivo, o *Mahout* é implementado sob o *Apache Hadoop* [2], que é uma plataforma que suporta processamento intensivo através da utilização de técnicas de *MapReduce* [13, 21, 24] e de processamento distribuído. Além desta vantagem, o *Mahout* inclui algoritmos de classificação e *clustering*.

Existem outras ferramentas utilizadas na extracção de conhecimento, bem como artigos que as comparam, como por exemplo em [1].

Na aplicação resultante desta dissertação, é utilizada a plataforma *WEKA*, escolhida por ser *open source*, pela documentação disponível *online*, e também pela utilização frequente no ambiente académico da Universidade do Minho.

2.8 Terminologia

Os termos utilizados para caracterizar cada uma das entidades participantes, apenas fazem sentido devidamente contextualizados. Apresenta-se, de forma sumária, o seu significado no contexto actual, bem como as suas propriedades mais importantes.

Eurotux - Ambiente empresarial no qual foi desenvolvido o sistema de classificações, e onde foram obtidos os dados utilizados para a escrita da dissertação.

Logs - Registos de eventos provenientes das aplicações instaladas nas máquinas, bem como do registo de eventos do sistema operativo.

Utilizador - Membro da equipa de administração de sistemas que utiliza a interface de classificação, quer para observar problemas derivados da classificação automática, como para classificar/validar *logs*.

Cliente - Entidade que possui serviços ou servidores geridos pela empresa Eurotux, que geram *logs* a analisar pelo sistema de classificação.

[9] *Waikato Environment for Knowledge Analysis.*

ETALC - Nome do sistema de classificação de *logs* implementado - *Eurotux Automatic Log Classification.*

etalcDB - Base de dados que armazena os *logs* e a respectiva classificação.

Interface de Classificação - Interface *web* desenvolvida para apresentar os *logs* ao utilizadores. É também através desta interface que são feitas as classificações.

Motor de Classificação - Aplicação que recolhe os *logs*, aplica regras de classificação, e que os regista na base de dados *etalcDB*.

Capítulo 3

Sistema de classificação de *logs*

Como descrito nas secções anteriores, este projecto de investigação e desenvolvimento visa a criação de uma plataforma de catalogação e classificação de *logs*. O presente capítulo começa por introduzir as principais entidades[1] existentes, e identifica as suas responsabilidades. Depois do contacto com a visão global da arquitectura, cada entidade é abordada com mais detalhe, sempre no contexto desta dissertação.

3.1 Arquitectura Global do Sistema

O sistema de classificação está dividido em dois sub-sistemas: o sistema que recolhe os *logs* e procede à sua classificação; e o sistema que apresenta os *logs* aos membros da equipa de administração de sistemas. Na Figura 3.1, os dois sub-sistemas são separados pela linha horizontal a tracejado. São eles: o motor; e a interface *web* de classificação. A comunicação entre os dois sub-sistemas é efectuada através da base de dados *etalcDB*.

Na Figura, também se pode observar quais as plataformas/linguagens de desenvolvimento utilizadas. Na concepção do motor é utilizada a linguagem *perl*, que possui nativamente mecanismos para o uso de expressões regulares. Considerando a natureza do problema, que se baseia no processamento em linhas de texto, é uma solução adequada. Para a implementação da interface *web*, é utilizada uma plataforma de desenvolvimento de aplicações *web*, *django*, escrita na linguagem *python*.

[1] Existem outras entidades que permitem a extensão da aplicação. Porém são soluções de implementação cujos detalhes não contribuem directamente para o assunto desta dissertação.

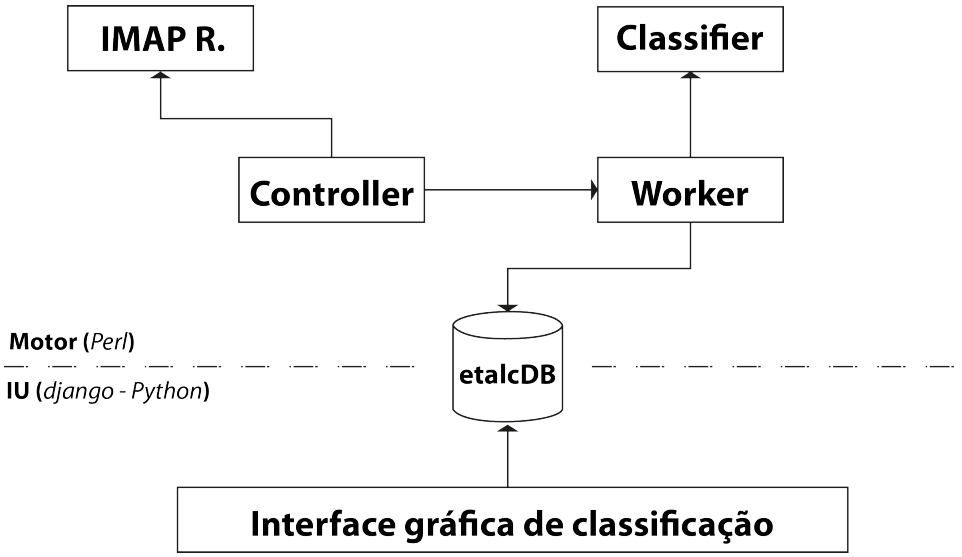


Figura 3.1: Arquitectura global do sistema

3.2 Agregação dos *logs*

Normalmente as notificações são efectuadas via email para o utilizador com privilégios de administração - *root* no caso de sistemas *Unix*. Para facilitar a agregação, os servidores são configurados para enviar os *logs/emails* para um ponto central, isto é, para uma caixa de correio.

Esta caixa de correio funciona como repositório de *logs* e, uma vez que suporta o protocolo IMAP de recepção de email, pode ser partilhada entre os vários membros da equipa de administração de sistemas. Após a criação deste repositório de *logs*, o administrador de sistemas pode consultar o estado dos seus servidores através de um cliente de email convencional. É a partir desta conta de email que o motor de classificação recolhe os *logs* a processar.

3.3 Motor de classificação

O motor de classificação é responsável pela recolha de dados, processamento, e registo de resultados na base de dados *etalcDB*. A recolha de dados implica copiar o *logs* ainda não processados, da caixa de correio.

O processamento, envolve a extracção das características do log (e.g. origem, data, assunto, conteúdo), a aplicação dos filtros estáticos, e à utilização dos mecanismos de classificação provenientes da mineração de dados, treinados com as classificações manuais.

Após a fase de processamento são registados os resultados, isto é, as características extraídas do *log* são inseridas na base de dados *etalcDB*, assim como a classificação prevista.

O motor de classificação possui alguns ficheiros de configuração, que o parametrizam. Estes ficheiros são carregados dinamicamente no decorrer da aplicação, pelo que pode ser alterados sem a necessidade de reiniciar a aplicação. Enumeram-se os ficheiros existentes:

1. *db.conf* - parametriza a conexão à base de dados *etalcDB*;
2. *filters.conf* - define os filtros estáticos a utilizar, abordados na Secção 3.5.1;
3. *imap.conf* - define as configurações de ligação à caixa de correio IMAP, e o tempo de espera entre recolhas;
4. *queues.cong* - define as directorias a utilizar no processamento, incluindo a directoria onde deve procurar novos *logs*;
5. *types.conf* - define os tipos de *logs* existentes.

3.3.1 Módulo *Controller*

A entidade *Controller* é o ponto de entrada do motor de classificação, sendo responsável pela interligação dos vários módulos e pelo fluxo de dados. Assim, auxiliado pelos ficheiros de configuração e pela base de dados *etalcDB*, procede aos seguintes passos:

1. Consulta a base de dados *etalc* de modo a identificar os clientes que estão activos;
2. Identifica quais são as directorias com *logs* de cada cliente;
3. Instancia o *IMAP Reader* para recolher os *logs* nas directorias encontradas;
4. Espera por notificações do *IMAP Reader*, que indicam se existem *logs* novos;
5. Caso existam *logs* por processar, instancia *Workers* que processam os *logs*.
6. Aguarda que os *Workers* terminem o seu trabalho;
7. Suspende a sua actividade durante o período de tempo definido no ficheiro de configuração.

Os passos anteriores são sequenciais e repetidos periodicamente. A redução do tempo de espera entre cada ciclo traduz-se numa pesquisa/-classificação de *logs* mais agressiva, ou seja, o tempo que existe desde que os *logs* chegam à caixa de correio até que surgem na interface de classificação é menor. Por outro lado esta decisão implica o aumento de trafego de rede, e de processamento por parte do servidor de correio e de classificação de *logs*.

3.3.2 Módulo *IMAP Reader*

A entidade *IMAP Reader* tem como responsabilidade recolher os *logs* a serem processados. É invocado pelo *Controller* com o intuito de receber novos *logs*. Abaixo, enumeram-se os principais passos que a entidade segue para a recolha de dados:

1. Recebe por parâmetro as directorias onde deve procurar *logs* novos;
2. Verifica qual o identificador[2] do último *log* recebido em cada directoria da caixa de correio[3];
3. Cria processos para fazer a recolha;
4. Inicia o processo de transferência;
5. Ao terminar, o processo guarda os *logs* no sistema de ficheiros local[4] e actualiza a estrutura de dados que mantém o identificador único do *log* em cada directoria.
6. Notifica o *Controller* de que tem *logs* por processar.

Existem abordagens mais eficazes, como por exemplo, quando o email dá entrada no servidor de correio electrónico é também enviado para o sistema de *logs*. Esta solução implicaria a criação de uma aplicação no servidor de email que envie os emails para o servidor de *logs*, ou ao redireccionamento dos emails para o servidor de email local à máquina de *logs*. Seria também necessário proceder à definição de regras de acesso na *firewall*.

É possível extender a plataforma de classificação/catalogação automática de *logs*, de modo a aceitar outras fontes de dados, bastando para isso implementar um *Reader*. Alternativamente, os *logs* podem ser colocados na directoria *pre_queue*, existente no sistema de ficheiros local, que é consultada periodicamente de modo a verificar se existem *logs* por processar.

3.3.3 Módulo *Worker*

O *Worker* representa a entidade que procede à interpretação do *log*, classifica-o e regista o resultado na base de dados *etalc*.

1. No inicio o *Worker* recebe a identificação de um cliente para o qual deve processar os *logs*;

[2]Os identificadores em questão são incrementais, e únicos em cada directoria.

[3]Mantém uma estrutura de dados de forma persistente, através do uso de mecanismos de persistência nativos do *perl*.

[4]Existe uma directoria de que mantém os *logs* a processar, chamada *pre_queue*.

2. De seguida consulta a base de dados *etalcDB* de modo a saber qual o último *log* classificado[5];

3. Identifica o tipo de *log*, com base no *subject* da mensagem, e no ficheiro de configuração que define os tipos existentes;

4. Carrega os filtros estáticos a aplicar, através de fábricas[6] de filtros;

5. Procede à classificação automática via filtros estáticos;

6. Caso a aplicação dos filtros não seja conclusiva, ou não hajam filtros para o seu tipo/cliente, são aplicados ao *logs* os mecanismos de classificação resultantes da mineração de dados;

7. Os *logs* e os resultados são inseridos na base de dados *etalcDB*.

Os passos para a classificação automática através de filtros estáticos são descritos na Secção 3.5.1. A aplicação métodos de classificação através de regras de mineração encontram-se na Secção 3.5.3;

3.3.4 Módulo *Classifier*

O *classifier* é a entidade que atribui a classificação aos *logs* submetidos pelos *Workers*. De modo a tornar possível a classificação, recorre aos filtros estáticos e às regras extraídas no processo de mineração de dados.

A aplicação de filtros baseia-se em expressões regulares aplicadas às várias linhas do *log*. Estas expressões regulares podem ser um conjunto de palavras que indiquem que não existe qualquer tipo de problema, por exemplo, se nos *logs* produzidos pelo *Arkeia* for encontrada uma linha com o texto *"job completed successfully !"*, então não existe qualquer problema. Deste modo, e de forma a possibilitar a definição genérica de filtros estáticos, é também possível definir filtros que identifiquem padrões que permitam classificar o conteúdo do *log* como OK.

Quando é encontrado um indicador de que o conteúdo do *log* não tem informação sobre problemas, deixa de ser necessário aplicar os restantes filtros, isto é, se há algo no conteúdo do *log* que deterministicamente nos diz que não há problemas, deixa de fazer sentido procurar por problemas. Considerando este facto, no motor de classificação implementado são aplicados os filtros "positivos"(que classificam o *log* como **OK**) antes dos filtros "negativos"(que classificam como **NOK**).

Apesar do processo de filtragem terminar quando é aplicado com sucesso um filtro positivo, o mesmo não acontece com os filtros que detectam problemas, uma vez que quantos mais problemas forem catalogados, maior será

[5]Este procedimento é apenas uma verificação adicional que evita *logs* duplicados na base de dados.

[6]Padrão de desenho de aplicações *Abstract Factory*.

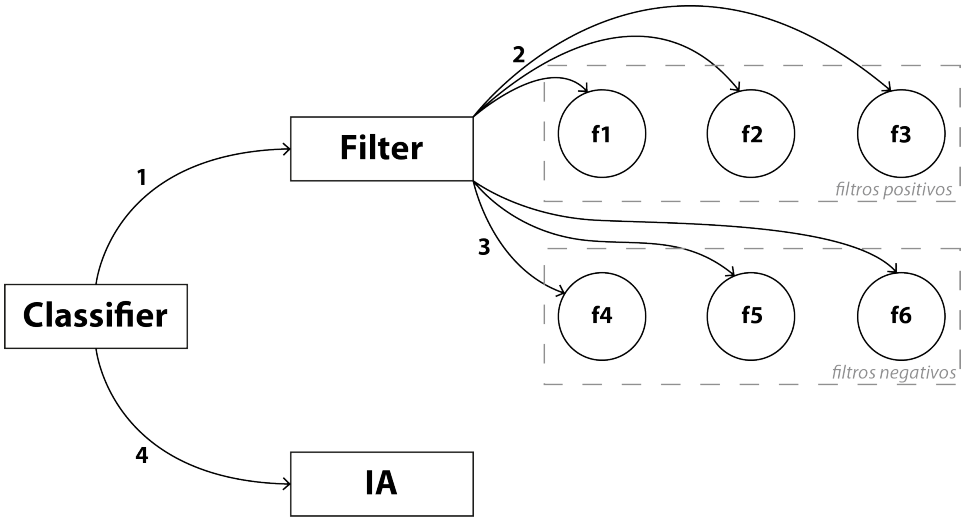


Figura 3.2: Arquitectura do Classificador

a ajuda prestada ao administrador de sistemas, e mais precisa será a sua classificação. Novamente, como o referido no início do capítulo, a precisão e qualidade das regras extraídas na mineração de dados advém da qualidade dos dados de treino, ou seja, das classificações feitas.

A Figura 3.2 ilustra o processo de aplicação de filtros. Os números da imagem correspondem aos seguintes passos:

1. O *Classifier* invoca a entidade *Filter*, responsável pela aplicação dos filtros estáticos.

2. *Filter* aplica os filtros "positivos"(*f1*, *f2* e *f3* na Figura 3.2), de acordo com a ordem estabelecida - ver Secção 3.5.2. Caso algum dos filtros consiga classificar o *log* como *OK*, o processo de classificação termina.

3. Caso contrário, são aplicados os filtros "negativos"(*f4*, *f5* e *f6* na Figura 3.2), que detectam os problemas.

4. Por último, é aplicado o modelo de classificação proveniente dos algoritmos inteligentes (IA na Figura 3.2).

3.4 Armazenamento de *logs*

A base de dados *etalcDB* que armazena os *logs* enfrenta alguns problemas de escalabilidade, nomeadamente no que diz respeito a espaço em disco utilizado. O espaço necessário ocupado por um *log* é variável, desde 4KB até aproximadamente 20MB. Por outro lado, o número de *logs* que dão

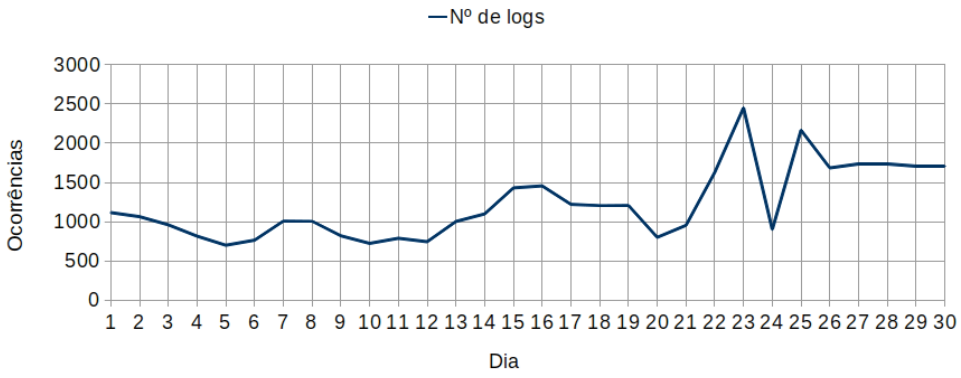


Figura 3.3: Variação do número de *logs* no mês de Setembro

entrada diariamente na caixa do correio também é elevado. A Figura 3.3 ilustra a variação do número de *logs* recebidos durante o mês de Setembro.

Para evitar o problema e tornar a aplicação escalável, é executada diariamente uma rotina de limpeza de *logs*, fora do horário laboral. A rotina segue o procedimento descrito abaixo, embora na implementação alguns dos passos sejam agrupados, por questões de optimização:

1. Identifica *logs* com idade superior a n dias[7];

2. Filtra os que não tenham sido classificados pelos utilizadores (por não conterem qualquer informação, a extrair no processo de mineração de dados);

3. Remove os *logs*;

4. Notifica o administrador da máquina da operação efectuadas via email.

Note-se que o conteúdo dos *logs* classificados continua armazenado, apesar de actualmente não contribuir para a extracção de conhecimento. No entanto não se exclui a futura utilização deste atributo.

3.5 Classificação automática

A seguintes secções descrevem o processo de classificação automática, quer através da definição de filtros estáticos, como através do modelo de classificação extraído da mineração de dados.

[7]Valor variável de acordo com a parametrização da rotina.

3.5.1 Definição de filtros estáticos

Na criação de um filtro é possível definir dois métodos: *check_ok* e *check_nok*. O primeiro método valida padrões positivos no conteúdo *log*, como por exemplo frases semelhantes a *"tarefa completada com sucesso"*. O segundo método procura por padrões de erros, e elege a linha do texto que melhor caracteriza o problema.

Porém a definição de filtros pode ser mais complexa do que tentar fazer coincidir padrões. Pode implicar seguir um conjunto de regras ou comportamentos, espectáveis. Para dar uma maior independência ao programador que desenvolve os filtros, é passado por parâmetro aos métodos o caminho e nome do ficheiro a analisar. Outra solução mais eficiente passaria por manter em memória o conteúdo do *log*, por exemplo, indicar no ficheiro de configuração *queues.conf*, que a directoria *pre_queue* se encontra em /dev/shm.

O retorno do método *check_ok* é um valor inteiro, que indica o sucesso do filtro. Já o retorno do método *check_nok* pode ser de um de dois tipos: o valor inteiro 0 (que representa o valor booleano *false*) quando não foi detectado o problema; ou um objecto do tipo *FilterRsp*. Este objecto contém os atributos que caracterizam o problema, indicando o nome do filtro que descobre a anomalia, o nome do serviço onde procurou o problema, o índice da linha de início e termino, a descrição do problema, a linha de texto que melhor o caracteriza, e o nível de criticidade. As Listagens 3.1 e 3.2 mostram a implementação do tipo *FilterRsp* e o modelo de implementação do método *check_nok*, respectivamente.

```
1  sub new{
2      my($class)  = shift;
3      my(%p) = @_;
4
5      bless {
6          'SRVNAME'     # Nome do servico (e.g. sshd)
7          'NAME'        # Nome do filtro
8          'CRITICALITY'# Criticidade do problema (1-5)
9          'STARTLINE'   # Indice da linha de inicio (opcional)
10         'ENDLINE'     # Indice da linha de termino (opcional)
11         'DESCRIPT'    # Descricao sobre o problema (opcional)
12         'TEXT'        # Texto da linha que melhor o caracteriza
13     }, $class;
14 }
```

Listagem 3.1: Object *FilterRsp*

```
1  sub check_nok{
2      my $filename = shift;
3      unless(open FILE, $filename){
4          warn "$!";
5          return  Filter::Response->new(
6              'SRVNAME'      => 'sshd',
7              'NAME'         => 'check_ssh',
```

```
8                 'CRITICALITY' => 5,
9                 'DESCRIPT'    => $!
10            );
11        };
12
13        ### Logica do filtro
14
15        ###
16
17        close FILE;
18        return 0;
19    }
```

Listagem 3.2: Método *check_nok*

3.5.2 Atribuição hierárquica de filtros

Os filtros nem sempre se ajustam a todos os clientes, e a todos os tipos de *log*. Por exemplo, um filtro desenhado especificamente para os *logs* do Logwatch, não deve ser aplicado aos *logs* do *Cron*. Isto porque, além de poderem classificar erroneamente o *log*, implicam processamento desnecessário para o servidor de classificação.

Tendo em consideração o problema anterior, surge a necessidade de poder atribuir filtros desenhados especificamente para determinado tipo de *log* ou cliente. Porém, é também requisito que se possa atribuir filtros de carácter genérico a um conjunto de clientes.

Existe para o efeito, o ficheiro de configuração *conf/filters.conf*, que possibilita a atribuição de filtros a clientes, bem como especificar o cliente do qual se pretende herdar a configuração de filtros. Esta segunda opção, permite criar uma estrutura hierárquica de filtros. Para terminar, por omissão, todos os clientes herdam os filtros do cliente virtual *default*.

```
1  ########## default ############
2  [default_arkeia]
3    0 = jobSuccess
4
5  [default_cron]
6    0 = zzz
7
8  [default_logwatch]
9    0 = diskspace
10
11 [default_nagios]
12   0 = state
13
14
15 ########## eurotux ############
16 [eurotux_arkeia]
17   parent = default
```

```
18
19 [eurotux_cron]
20   parent = default
21
22 [eurotux_logwatch]
23   parent = default
24   0 = http
25   1 = sshd
```

Listagem 3.3: Exemplo de atribuição de filtros

A listagem 3.3 ilustra uma possível configuração do sistema de filtros. Note-se que está presente a definição hierárquica de filtros, onde *default* é o nó raíz. Este nó é precedido do nó *Eurotux* que herda os seus filtros[8]. Neste exemplo, são também acrescentados dois filtros aos *logs* do *Logwatch*, *http* e *sshd*. Os valores **0** e **1** indicam a ordem pela qual devem ser aplicados.

3.5.3 Mineração de dados

A Secção 2.5.2 refere de forma genérica a abordagem necessária para a extracção de conhecimento. Refere-se que é necessário extrair as variáveis - ou atributos - que contribuem directa ou indirectamente para a classificação do *log*. Apesar de estarmos perante um problema de classificação binário, onde o *log* pode ser catalogado como OK ou NOK, é possível dividir a categoria dos NOK em níveis de problema - a criticidade do problema. Neste estudo de caso, considera-se que os *logs* cujo conteúdo não evidencia problemas, possuem o nível de criticidade 0, e que os restantes variam entre os valores 1 e 5.

Deste modo, os administradores de sistemas que utilizam o sistema de classificação, podem escalonar a resolução de problemas de acordo com o que consideram mais importante.

Selecção e pré-processamento de atributos

Há uma colecção de atributos que caracterizam o *log*. Porém nem todos contribuem com informação relevante para o treino dos algoritmos de classificação. Por exemplo, o atributo *id* é um valor sequencial, incrementado de cada vez que um *log* dá entrada na base de dados. Ao analisar este atributo, o algoritmo inteligente tenta achar uma relação entre o seu valor, e a classificação atribuída. Porém, não se pode inferir a classificação do *log* através deste atributo, uma vez que são completamente independentes.

A tabela 3.1 mostra os atributos seleccionados, o tipo de dados como são tratados, se admitem valores nulos, e caso admitam o valor pelo qual é substituído. O texto a negrito na última linha da tabela, mostra o atributo classe, isto é, o atributo a inferir através da análise dos restantes.

[8] Uma vez que no ficheiro de configuração o seu *parent* é o nó *default*.

Tabela 3.1: Atributos seleccionados para a mineração de dados

Nome	Tipo	Admite nulos	Valor neutro
data	date	não	-
origem	nominal	não	-
destino	nominal	não	-
assunto	nominal	não	-
tipo de conteúdo	nominal	não	-
tipo de *log*	nominal	não	-
cliente	nominal	não	-
nº total de linhas	numérico	não	-
linha de erro	nominal	sim	*string* vazia
criticidade	**nominal**	**sim**	**0**

Quando um *log* novo é recolhido pelo motor de classificação, não possui os atributos *linha de erro* e *criticidade* preenchidos. Estes atributos são preenchidos através de um de dois modos: ou através dos mecanismos de classificação automática; ou através do preenchimento manualmente efectuado pelo utilizador.

Geração Dinâmica do Modelo de Classificação

O modelo de previsão é gerado através das classificações efectuadas manualmente, ou na validação explícita das classificações efectuadas automaticamente. No entanto, o tipo e formato dos *logs* não é estático, variando com actualizações de *software*, configuração de serviços das máquinas, entre outros.

Considere-se um modelo gerado a dada altura, ao qual é aplicado de um conjunto de dados de teste derivado das classificações efectuadas aos *logs* existentes nessa data, e que se obtinha 75% de instâncias classificadas correctamente. Com a alteração do tipo e formato dos *logs*, esta percentagem tenderia a ser menor, pelo que o modelo deveria de ser recalculado.

Isto significa que o processo de classificação/validação de *logs* manual não deixa de ser necessário, embora seja cada vez mais reduzido uma vez que o modelo de classificação passa a contemplar mais casos.

O motor possibilita recalcular o modelo de previsão. Utiliza para o efeito os algoritmos existentes na plataforma de mineração de dados *WEKA*, e a base de dados *etalcDB* que contém o histórico das classificações. Os passos abaixo descrevem o procedimento[9]:

1. Invoca a *stored procedure* que agrega os *logs*, os respectivos problemas e filtra valores nulos;

[9]Este procedimento representa as fases de selecção, pré-processamento e de transformação, referidas na Secção 2.5.2.

2. Cataloga na base de dados *etalcDB* as linhas de erro detectadas no processo de classificação manual, bem como o número de ocorrências. A tabela gerada é consultada quando surgem novos *logs*, de modo a tentar preencher o atributo *linha de erro*, essencial para a sua classificação.

3. Aplica regras de normalização, nomeadamente passar os atributos numéricos para valores nominais;

4. Treina o algoritmo de classificação pretendido com os dados recolhidos;

5. Remove dados de carácter temporário gerados no processo.

Nem todos os *logs* possuem problemas. Nesse caso o atributo classe *criticidade* é preenchido com o valor 0 e o atributo *linha de erro* com a linha vazia (sem caracteres). É também de anotar que, se foram identificados dois problemas no *log*, serão criados dois registos semelhantes para treino do algoritmo onde apenas variam os atributos *criticidade* e *linha de erro*.

3.5.4 Utilização do modelo de classificação

Quando surge um *log* na caixa de correio, o motor tenta identificar os problemas existentes e atribuir-lhe uma classificação.

De forma a conseguir encontrar os problemas do *log*, existe um processador de texto que percorre as suas linhas, tentando identificar os erros. Para o efeito, o processador consulta a tabela da base de dados *etalcDB* que contém os erros conhecidos (criada no processo de geração do modelo). Esta tabela tem como origem as classificações efectuadas pelos utilizadores.

Por cada problema detectado é criado temporariamente no motor um registo com os atributos do *log* e com a linha de erro (atributo *linha de erro*). Depois, cada registo é submetido ao modelo de classificação gerado, através do qual se obtém a criticidade do problema e o grau de confiança desta classificação. Por último, os resultados são inseridos na base de dados, para que possam ser observados pelos utilizadores.

3.6 Interface de classificação

A interacção entre os utilizadores e os *logs* é feita através da interface de classificação. De acordo com o especificado na Secção 2.4, o conjunto de *logs* é distribuído pelos utilizadores de modo a que sejam vistos o maior número de *logs*, optimizando os recursos e evitando que vários elementos vejam o mesmo *log*. Assim, quando o utilizador se autentica na aplicação, são-lhe apresentados os clientes para o quais deve ver *logs*, como ilustra a Figura 3.4.

Quando o utilizador selecciona um dos clientes, é-lhe apresentada uma lista de *logs*, Figura 3.5. Nas colunas da tabela encontram-se os principais atributos que caracterizam o *log*. A coluna *classification* indica a classificação atribuída pelos mecanismos automáticos. No lado direito são apresentados alguns filtros que facilitam a pesquisa e, sob a tabela, é possível pesquisar *logs* pelos seus atributos.

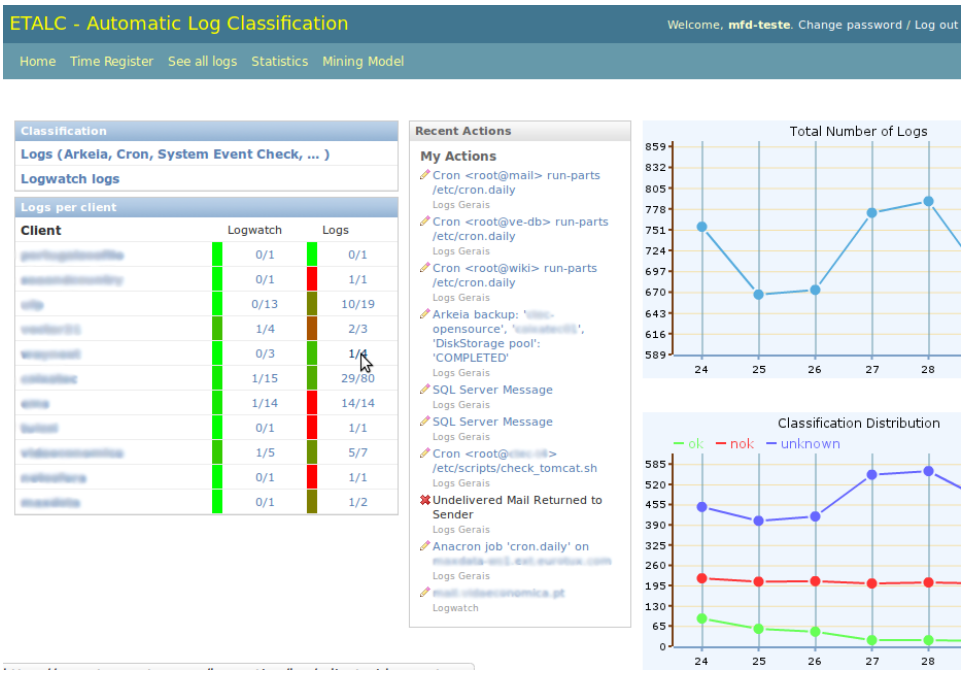


Figura 3.4: Vista de *logs* diários

3.6.1 Processo de classificação e validação manual

A classificação efectuada pelos elementos da equipa de administração de sistemas é de extrema importância para o mecanismo de classificação automática. Para treinar adequadamente os algoritmos de extracção de conhecimento, são precisas classificações correctas, impessoais e em grande número. Deste modo é requisito que a interface gráfica permita, sem grande esforço, que os utilizadores classifiquem *logs*.

Ao acompanhar os utilizadores, e sendo o processo de análise e classificação de *logs* um processo moroso, conclui-se que o número de interacções necessárias para efectuar a classificação deve ser o mais reduzido possível. Na tentativa de minimizar o número de interacções, o formulário é apresentado com os campos previamente preenchidos, isto é, com os atributos que caracterizam *log* - conteúdo, origem, data, etc - e com os atributos derivados da classificação automática - criticidade e linha de erro.

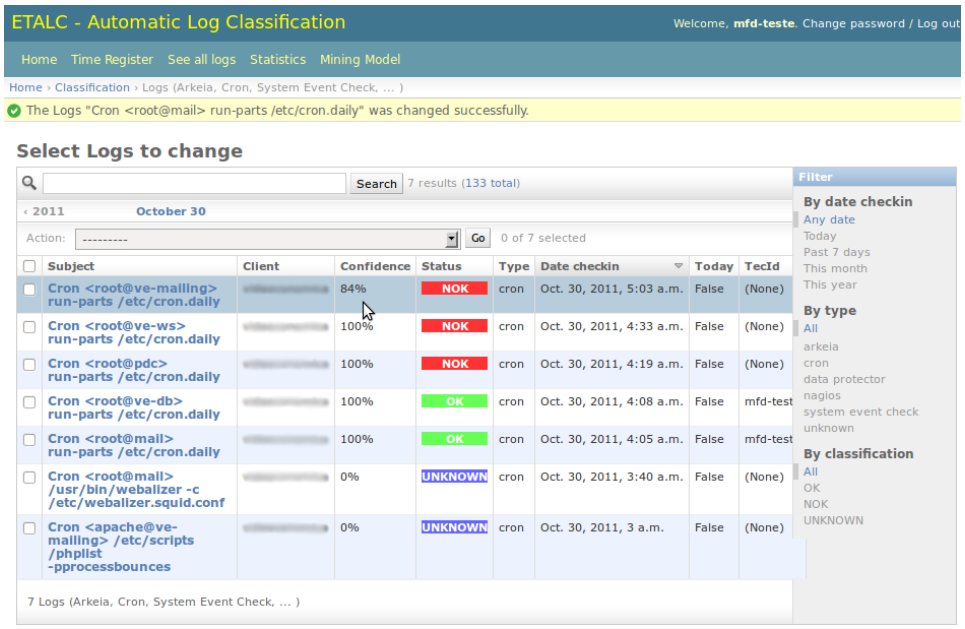


Figura 3.5: Lista de *logs*

O processo de classificação nem sempre é trivial, pois *é pedido aos utilizadores que identifiquem a linha que melhor caracteriza o problema*, o que nem sempre é possível. Um determinado problema pode conter várias linhas importantes, o que leva o utilizador a escolher entre uma das linhas. Assim, dois *logs* que apresentem o mesmo conteúdo podem ser classificados de forma diferente. Felizmente, nos *browsers* da actualidade é apresentada uma sugestão de preenchimento da linha de erro, baseada em selecções previamente efectuadas. Isto minimiza o problema e estimula o utilizador a efectuar classificações de modo determinístico.

3.6.2 Registo no *Request Tracker* e estatísticas

Todos os dias os utilizadores fazem classificações via interface *web*. De modo a controlar o tempo necessário para a análise dos *logs*, e a poder justificar a factura apresentada ao cliente, a aplicação mantém o registo das classificações efectuadas por cada utilizador.

Após a utilização pretendida, o utilizador selecciona a opção *registo de tempo*. É apresentado um formulário que solicita a introdução do tempo despendido na análise dos *logs*. No processo de submissão, é enviado para o RT - *Request Tracker* - o tempo preenchido e o resumo das classificações (ou da validação de classificações automáticas).

Como são registadas as interacções feitas, o utilizador sente-se estimulado e motivado a fazer novas classificações.

Na interface de classificação são também disponíveis gráficos que permitem visualizar o estado do sistema. É possível analisar o número de *logs* que chegam diariamente, a sua distribuição pelos clientes, e a quantidade de *logs* que foram classificados pelo mecanismo automático.

Capítulo 4

Resultados

Este capítulo dedica-se à análise dos resultados obtidos. Começa por explorar as classificações feitas através da interface *web*, seguindo-se da analise ao resultado das classificações automáticas.

4.1 Análise das classificações manuais

Os dados obtidos pelas classificações feitas pelos utilizadores, não estão livres de erros. Estes devem ser minimizados de forma a não prejudicar os mecanismos de classificação automática. Assim, a correcta implementação da interface *web* torna-se fundamental.

Sabemos que é impossível passar a produção uma versão de *software* livre de problemas, sendo sempre necessário corrigir erros que surjam na sua utilização. O caso da aplicação ETALC não foi diferente, e apareceram erros que poderiam prejudicar o treino dos algoritmos de classificação. Na verdade, a primeira versão o ETALC não validava correctamente os campos a preencher pelos utilizadores, podendo os *logs* ser classificados como problemáticos sem ser necessário introduzir o campo com a linha de erro. Isto levou a que na fase de pré-processamento de dados, descrita na Secção 2.5.2, fosse necessário ignorar algumas das classificações.

Foi encontrado outro problema relacionado com a selecção da linha de erro. Os utilizadores ao seleccionar a linha, por vezes, seleccionavam também o carácter de mudança de linha. Isto é, em alguns casos obtinha-se *"linha de erro"*, noutros *"linha de erro\n"*, o que é errado uma vez que ilude o sistema com dois problemas que na realidade são o mesmo. Neste caso a alteração foi ao nível da aplicação *web*, onde a linha passou a ser pós-processada pelo servidor.

Outra conclusão tirada da análise das classificações é a sua distribuição pelos vários tipos de *log*. Analisando o gráfico da Figura 4.1, observa-se que os *logs* do *Logwatch* são mais classificados, o que é um indicador de que são mais importantes, ou os utilizadores têm maior facilidade em classifica-los.

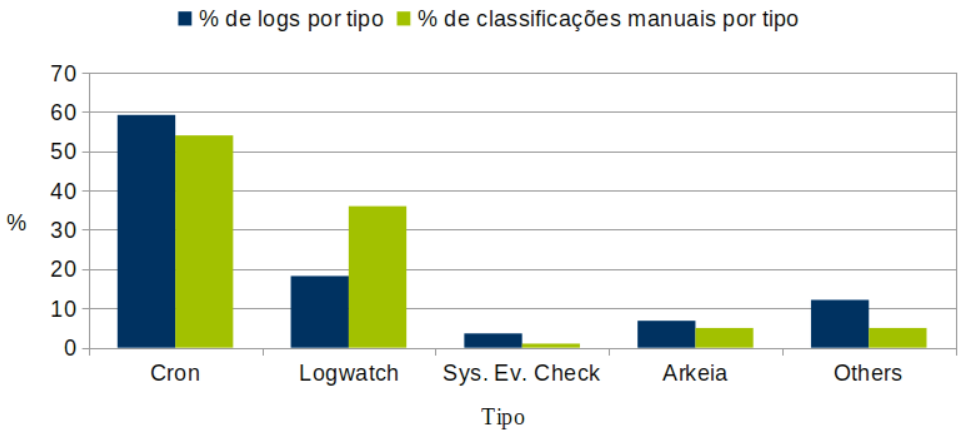


Figura 4.1: Comparação entre os *logs* existentes e as classificações recolhidas

Por último, o gráfico 4.2 ilustra o número de ocorrências das várias linhas de erro seleccionadas[1]. Da análise do gráfico podemos concluir que há problemas que são recorrentes, ou que demoram a ser resolvidos.

Figura 4.2: Número de ocorrências das linhas de erro

4.2 Análise das classificações automáticas

Durante o utilização da interface *web*, foram recolhidos 58105 *logs*, de entre os quais se obtiveram 1958 classificações manuais.

[1]Não foram consideradas linhas de erro com apenas uma ocorrência

No presente, estão em uso quatros filtros estáticos que classificam cerca de 7% dos *logs*. A principal razão pela qual não existem mais filtros, deve-se ao facto da lógica necessária à sua concepção não ser trivial. A implementação por si não é complicada, mas é necessário ter em atenção o ambiente em constante mutação.

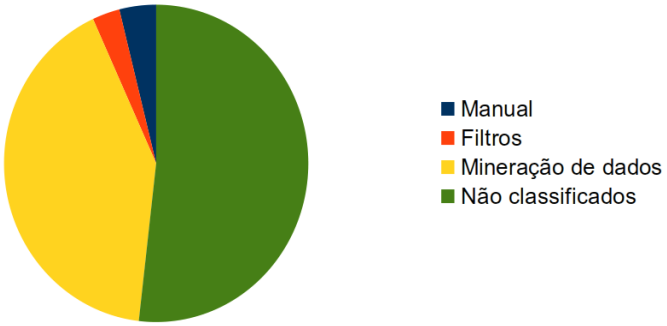


Figura 4.3: Classificações obtidas

A imagem da Figura 4.3 ilustra as classificações obtidas. O sistema criado classifica aproximadamente 44% dos *logs*. As classificações efectuadas pelos utilizadores representam cerca de 4%, e o número de *logs* por classificar é aproximadamente 52%.

4.3 Estatísticas

Os gráficos estatísticos existentes na interface gráfica de classificação também contribuíram para analisar o sistema. Recordando o gráfico da Figura 3.3, observa-se que existiu número de *logs* fora do normal - nos dias 22, 23 e 25 de Setembro. Neste caso foi necessário reconfigurar alguns servidores de modo a corrigir o problema. Isto sugere que poderia ser acrescentada uma funcionalidade que verificasse periodicamente o número de *logs*, que caso fosse encontrado uma frequência anómala avisasse os administradores de sistema.

Ainda neste problema, o gráfico com a distribuição do número de *logs* por cliente foi uma ajuda, facilitando o processo de descoberta do problema.

Capítulo 5

Conclusões

Ter noção do estado das infra-estruturas que suportam os serviços é fundamental, de modo a que quando surjam problemas sejam resolvidos o mais rapidamente possível. Normalmente, são associados à infra-estrutura sistemas de monitorização - e.g. *Cacti* e *Nagios* - que na presença de problemas notificam os membros da equipa de administração de sistemas. Porém, estes sistemas são genéricos e apenas monitorizam o conjunto típico de serviços utilizados nos servidores. Deste modo, é necessário classificar os registos provenientes de aplicações específicas.

Considerando o conjunto de classificações recolhidas, apenas 6,4% dos logs constituem anomalias de sistemas, sendo ideal que apenas estes fossem vistos. Deste modo, surge a necessidade de criar um mecanismo de classificação automática.

Através da análise do conjunto de dados utilizado, conclui-se que existem alguns tipos de *logs* que constituem a maioria. Estes, merecem atenção redobrada no sentido de os classificar, eliminando o maior número de *logs* a serem vistos.

Para criar um sistema de classificação automática podem seguir-se várias metodologias, como a definição de filtros estáticos recorrendo a expressões regulares, à definição de modelos estatísticos de Markov, ou ainda à mineração de dados. No último caso, podem ser utilizados vários algoritmos de classificação existentes - *Decision Trees*, *Naive Bayes*, *Bayes Nets*, *Support Vector Machines*.

As técnicas de extracção de atributos, utilizadas recorrentemente em sistemas de classificação de documentos, não solucionam o problema. Este processo de selecção é fundamental, e deve ser efectuado acompanhando os elementos da equipa de administração de sistemas no processo de classificação manual.

Existem ferramentas como o *WEKA*, *Apache Mahout* e o *RapidMiner* que facilitam o desenvolvimento de sistemas de classificação de *logs*, disponibilizando a implementação de vários algoritmos de mineração de dados.

A aplicação resultante desta dissertação está actualmente em uso na empresa Eurotux e tem se mostrado eficaz, classificando automaticamente cerca de 44% dos *logs* recebidos. No futuro, com a definição de mais filtros e com a recolha de mais classificações manuais, este número tenderá a ser superior.

O modelo de previsão gerado no processo de mineração de dados é dinâmico. Semanalmente é substituído, e conta com novas classificações. Esta característica confere à aplicação o caracter genérico necessário para a adaptação a novos tipos de *logs*.

Terminando, e ficando além do âmbito desta dissertação, a definição de meta informação sobre os *logs* facilitaria o desenho de aplicações de classificação, e contribuiria para a melhor comunicação entre o programador e o administrador de sistemas.

5.1 Principais problemas encontrados

Convencer os utilizadores a classificar *logs* não é uma tarefa simples, uma vez que o processo de classificação é repetitivo e moroso. O facto de ser repetitivo promove erros de classificação em que o utilizador assume que não existem problemas (por o *log* não ter tido problemas nas últimas ocorrências). Soluções para tentar minimizar o problema foram implementadas, como por exemplo a formação de sub-grupos de utilizadores que trocam diariamente o conjunto de *logs* a classificar.

Para minimizar o problema do processo ser demorado, e de forma a melhorar a usabilidade, foram tidos em consideração alguns padrões de desenho para interfaces gráficas. Nomeadamente os padrões *Good Defaults*, *Calendar Picker*, *Blank Slate*, foram utilizados no desenho do formulários de recolha de dados, *Breadcrumbs*, *Home Link*, *Pagination* para a navegação entre menus, *Live Filter*, *Sort By Column*, *Alternating Row Colors* para melhorar a pesquisa de *logs*, e ainda outros padrões como o *Completeness meter*. A descrição dos padrões de desenho referidos pode ser encontrada em [31]. Sempre que possível, foram pré-preenchidos os campos que o utilizador teria que preencher, quer com as classificações automáticas, quer através de pressupostos de que o *log* não tem problema (nos casos em que a classificação automática não está disponível).

A quantidade de literatura relacionada com o tópico do registo de eventos é escassa. Foram encontrados alguns artigos que, igualmente lamentavam a falta de informação [6, 10, 23].

5.2 Trabalho futuro

O modelo gerado actualmente pode ser refinado. Actualmente não são utilizadas técnicas que desviam o modelo de classificação, de modo a alterar

a percentagem de falsos positivos e de falsos negativos. Esta alteração é fundamental, uma vez que o aparecimento de falsos negativos é considerada como falha[1].

Da análise às linhas de erro seleccionadas pelos utilizadores, observa-se que para dado problema existem linhas semelhantes de erro. A construção de um analisador de texto que codifique as várias linhas numa linha genérica, representante do problema, é fundamental. O modelo de previsão tornar-se--ia mais genérico, e como consequência conseguiria classificar mais casos.

Como referido na Secção 3.5.3, é utilizada a plataforma de extracção de conhecimento *WEKA*. A implementação do motor de classificações permite seleccionar de entre os vários algoritmos existentes na plataforma, assim, como trabalho futuro devem ser efectuados testes no sentido de eleger o algoritmo que melhor se adequa ao problema.

[1]Os falsos negativos, neste caso, representam a atribuição de classificações com grau inferior à do problema real.

Bibliografia

[1] Dean W. Abbott, I. Philip Matkovsky, John F. Elder, and IV. An evaluation of high-end data mining tools for fraud detection. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 12–14. IEEE, 1998.

[2] Apache. Apache hadoop project. `http://hadoop.apache.org`, 2008.

[3] Apache. What is apache mahout? `http://mahout.apache.org/`, 2008.

[4] Arkeia. Arkeia network backup product suite. `http://www.arkeia.com`, 2010.

[5] Ian Berry, Tony Roman, Larry Adams, J.P. Pasnak, Jimmy Conner, Reinhard Scheck, and Andreas Braun. *Cacti Manual 0.8.7*, 2010.

[6] M. F. Buckley and D. P. Siewiorek. Vax/vms event monitoring and analysis. In *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, FTCS '95, pages 414–, Washington, DC, USA, 1995. IEEE Computer Society.

[7] P. Cobley, L. Jansz, and R. Appignanesi. *Introducing semiotics*. Totem Books, 1997. ISBN 9781874166559.

[8] cogNiTioN. *Intro to cron*, 1999.

[9] Susan Dumais, John Platt, Mehran Sahami, and David Heckerman. Inductive learning algorithms and representations for text categorization. pages 148–155. ACM Press, 1998.

[10] Ro Etalle, Fabio Massacci, and Artsiom Yautsiukhin. The meaning of logs. In *Proc. 4th Int. Conference on Trust, Privacy and Security in Digital Business (TrustBus*. Springer, 2007.

[11] EventReporter. Eventreporter - windows event monitoring and forwarding. `http://www.eventreporter.com/en/`, 2011.

[12] Internet Engineering Task Force. Rfc 5424: The syslog protocol, 2009.

[13] Dan Gillick, Arlo Faria, and John Denero. Mapreduce: Distributed computing for machine learning, 2006.

[14] Nabil Hammoud. Decentralized log event correlation architecture. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, MEDES '09, pages 79:480–79:482, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-829-2. doi: http://doi.acm.org/10.1145/1643823.1643919.

[15] Robert Kehl. *Cronw - cron for windows*, 2010.

[16] Zhenmin Li, Jed Taylor, Elizabeth Partridge, Yuanyuan Zhou, William Yurcik, Cristina Abad, James J. Barlow, and Jeff Rosendale. Uclog: A unified, correlated logging architecture for intrusion detection. In *IN THE 12TH INTERNATIONAL CONFERENCE ON TELECOMMUNICATION SYSTEMS - MODELING AND ANALYSIS (ICTSM*, 2004.

[17] Logcheck. Logcheck. `http://logcheck.org/`, 2011.

[18] Logwatch. Logwatch. `www.logwatch.org`, 2010.

[19] A Mccallum and Kamal Nigam. A comparison of event models for naive bayes text classification. *Dimension Contemporary German Arts And Letters*, 752:41–48, 1998.

[20] Core Development Team Nagios and Community Contributors. *Nagios Core Version 3.x Documentation*, 08 2010.

[21] Andrew Y. Ng, Gary Bradski, Cheng-Tao Chu, Kunle Olukotun, Sang Kyun Kim, Yi-An Lin, and YuanYuan Yu. Map-reduce for machine learning on multicore. In *NIPS*, 12/2006 2006.

[22] Nimsoft. `www.nimsoft.com`, 2011.

[23] Paul Radford, Andy Linton, and Ian Welch. Event log messages as a human interface, or, "do you pine for the days when men were men and wrote their own device drivers?". In *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction*, OZCHI '10, pages 160–163, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0502-0. doi: http://doi.acm.org/10.1145/1952222.1952253.

[24] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *In HPCA '07: Proceedings of the 13th International Symposium on High-Performance Computer Architecture*, pages 13–24. IEEE Computer Society, 2007.

[25] Rapid-I. Rapidminer. `http://rapid-i.com/`, 2011.

[26] Rsync. *rsync - a fast, versatile, remote (and local) file-copying tool*, 2011.

[27] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840.

[28] Manuel Filipe Santos and Carla Sousa Azevedo. *Data Mining: Descoberta De Conhecimento Em Bases De Dados*. FCA, 2005. ISBN 978-972-722-509-5.

[29] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[30] Splunk. `www.splunk.com`, 2011.

[31] UIPatterns. User interface design pattern library. `http://ui-patterns.com/patterns`, 2011.

[32] Jesse Vincent, Dave Rolsky, Darren Chamberlain, Richard Foley, and Robert Spier. *RT Essentials*. O'Reilly Media, Inc., 2005. ISBN 0596006683.

[33] Warren Weaver. *Recent Contributions to the Mathematical Theory of Communication*. 1949.

[34] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0120884070.

[35] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-486-3.